Introdução à Física Computacional

I. F. F. dos SANTOS

Maceió, 26 de outubro de 2021

Introdução à Física Computacional

I. F. F. dos SANTOS Versão 1.1

Esta apostila foi escrita usando LaTeX.
O código fonte pode ser encontrado em
<https://github.com/ismaeldamiao/Apostila_de_IFC>

Sumário

1. Introdução

QUANDO queremos dividir a física em ramos, para facilitar a sua compreensão, geralmente pensamos em dois principais ramos, a saber, Física Teórica e Física Experimental; Você deve ter estudado um pouco sobre ambos mas há ainda outro ramo que não pode ser encaixado nesses dois e é a Física Computacional, ela por sua vez pode ser organizada em outros quatro ramos 1) modelagem: Tanto a Física Teórica quanto a Computacional usam modelagem, os físicos buscam descrever os fenômenos da natureza utilizando modelagem matemática. 2) Instrumentação: Tanto a Física Experimental quanto a Computacional buscam desenvolver e utilizar equipamentos sofisticados, o que inclui aparatos de alto custo e criação de novas tecnologias, acontece que os experimentais usam os equipamentos para medir, ou seja para saber o que acontece quando a natureza interage com o aparato e daí fazer a coleta de dados enquanto que os computacionais usam esses recursos para implementar algoritmos de computador. 3) Tratamento de dados: Tanto a Física Experimental quanto a Computacional realizam coleta de dados e se faz necessário o uso de diversos meios para análise e interpretação desse dados. 4) Simulação: Especialidade da Física Computacional, uma vez que se possui um modelo teórico então chega o momento de simular um experimento físico.

Uma simulação é como um mini universo cujo as leis da física são justamente as do modelo teórico, é a simulação de um experimento que produz os dados a serem analisados mas, diferente da Física Experimental que quer verificar se a natureza de fato obedece o modelo, na Física Computacional o modelo sempre é obedecido pela simulação (exceto em erros de programação) o que é uma diferença notável. Apesar de que a metodologia de simulação lembra a de experimentação e de que a análise de dados de um experimento e de uma simulação segue basicamente os mesmos princípios, as conclusões que uma simulação fornece não têm tanta importância quanto as conclusões que um experimento fornece, tal importância continua no patamar especulativo, característica comum entre os trabalhos teóricos.

Os conhecimentos obtidos no estudo da computação lhe permitirão resolver equações algébricas e diferenciais, calcular derivadas e integrais, realizar transformadas de Fourrier e Laplace, fazer análise estatística de dados, etc, além de lhe proporcionar base para outras atividades como escrever documentos em LaTeX, escrever scripts para melhor gerenciamento do seu computador, etc.

O objetivo deste nivelamento é incentivar a estudar a programação antes mesmo de se deparar com as matérias a isso dedicadas, assim o choque não será tão grande quando chegar na matéria.

Neste módulo introdutório iremos aprender sobre algoritmos, pseudocódigos e fluxogramas, fazer uma introdução à linguagens de programação, conhecer o programa *Ola Mundo* de algumas linguagens e ver algumas aplicações básicas da programação para os primeiros períodos do curso de física.

2. Prelúdio aos Algoritmos

"Como você os ensina alguma coisa nova? Misturando o que eles sabem com o que eles não sabem" já dizia Picasso e é justamente essa a filosofia de um algoritmo, um algoritmo é uma série de instruções sobre como fazer alguma coisa e essas instruções devem estar divididas em pequenas partes simples que, quando seguidas todas, levam a execução de uma tarefa complexa.

Por exemplo, calcular o fatorial de um número natural não é uma tarefa simples pois está além das quatro operações básicas e se você quiser ensinar alguém a calcular o fatorial então deverá dividir essa tarefa em outras tarefas mais simples. Comece por observar a definição do fatorial.

$$n! = n \cdot (n-1)! = \prod_{i=1}^{n} i = 1 \cdot 2 \cdot ... \cdot n \; \forall \, n \in \mathbb{Z}$$

Foque na parte $n! = \prod_{i=1}^{n} i$, uma série de instruções simples para calcular o fatorial de n seriam:

1. Calcule $1 \cdot 2$.
2. $n = 2$? Se sim então $n! = 1 \cdot 2$. Senão então calcule $x_3 = 2 \cdot 3$.
3. $n = 3$? Se sim então $n! = x_3$. Senão então calcule $x_4 = x_3 \cdot 4$.
4. $n = 4$? Se sim então $n! = x_4$. Senão então calcule $x_5 = x_4 \cdot 5$.
5. ...
6. $n = x_i$? Se sim então $n! = x_i$. Senão então calcule $x_{i+1} = x_i \cdot (i+1)$.

Por outro lado, da parte $n! = n \cdot (n-1)!$ vem um algoritmo mais simples.

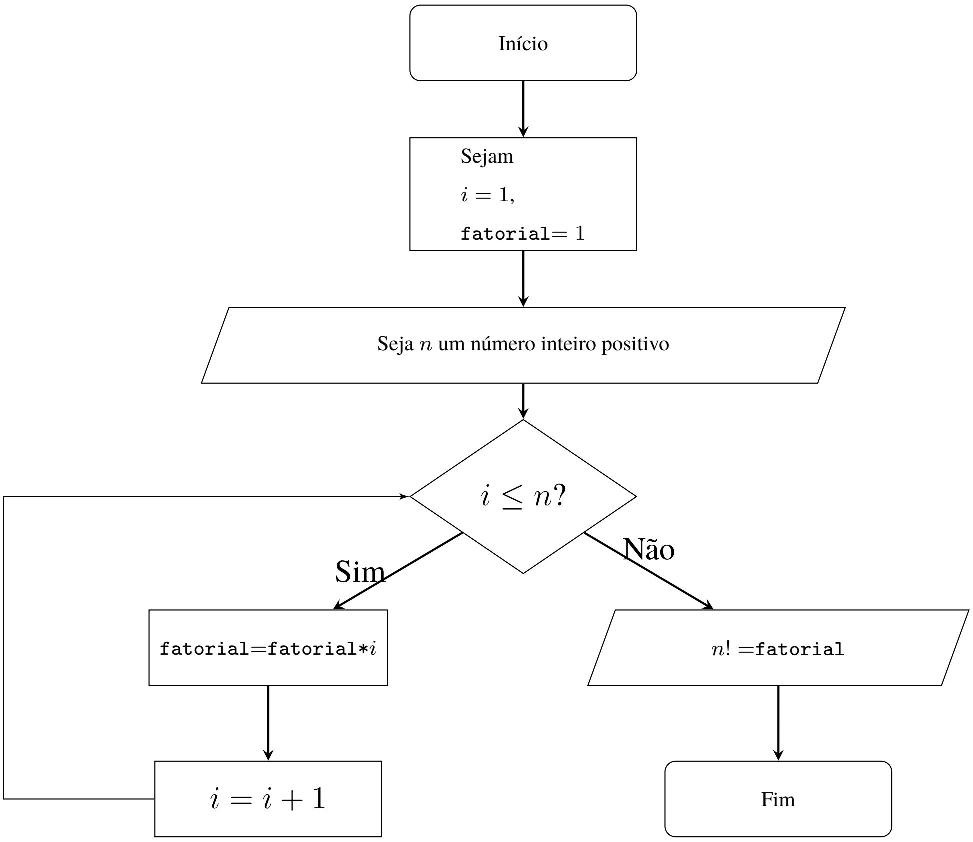
1. $n = 1$? Se sim então $n! = 1$. Senão então calcule $n \cdot (n-1)!$

Nesse último caso fica a pergunta: Como calcular $(n-1)!$? mas a resposta é simples, basta começar o algoritmo de novo e testar se $(n-1) = 1$, ou seja, o algoritmo é autorreferente, se o procedimento ainda não é o suficiente simples então você deve começar novamente o algoritmo, e novamente... até que o procedimento seja suficientemente simples. Este tipo de algoritmo é chamado **recursivo**, enquanto que o tipo anterior é chamado **iterativo**, obviamente há outros tipos de algoritmos.

Há duas principais maneiras de representar algoritmos, a saber, usando pseudocódigo ou fluxogramas. Vejamos como as usar.

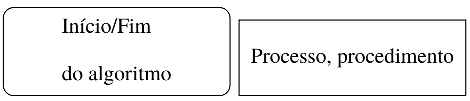
2.1. Fluxograma

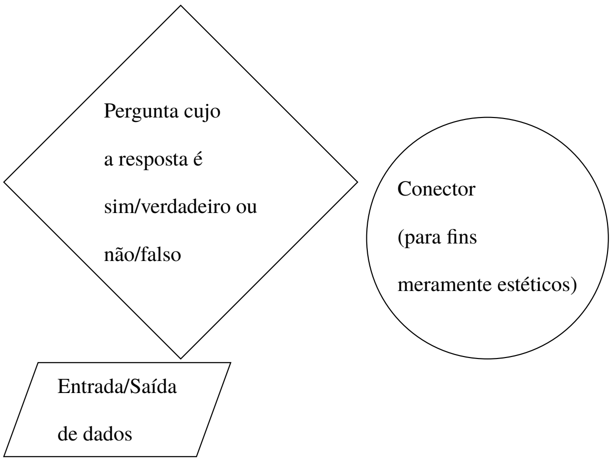
Um fluxograma é uma figura que exemplifica um algoritmo de forma simples e intuitiva. Por exemplo, o fluxograma abaixo mostra como calcular o fatorial de um número.



Pergunta 1. *Há dois cálculos desnecessários nesse algoritmo. Você consegue identificá-los? Como o algoritmo pode ser melhorado?*

No fluxograma há alguns símbolos mas não é preciso conhecer os símbolos para o entender pois ele é bastante intuitivo. Basicamente, os elementos de um fluxograma são

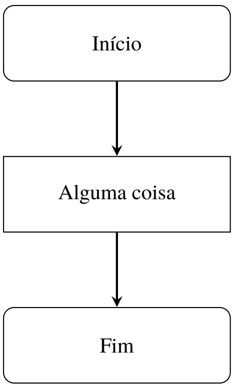




2.2. Pseudocódigo

Para explicar pseudocódigo irei recorrer à sua inteligência e aos fluxogramas.

Programa

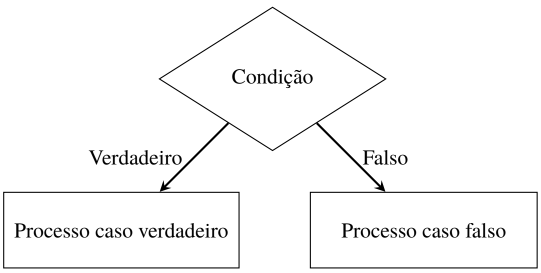



```
inicio

alguma_coisa()

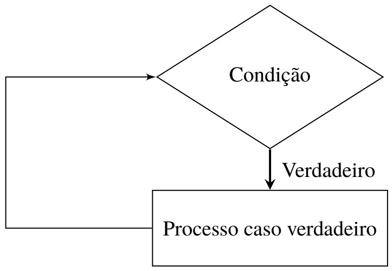
fim
```

Decisão



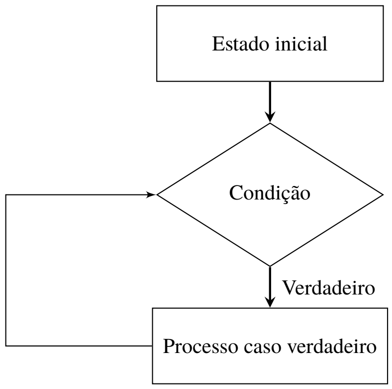

```
se condição entao
   Processo_caso_verdadeiro()
senao
   Processo_caso_falso()
fim se
```

Ciclo “Enquanto”



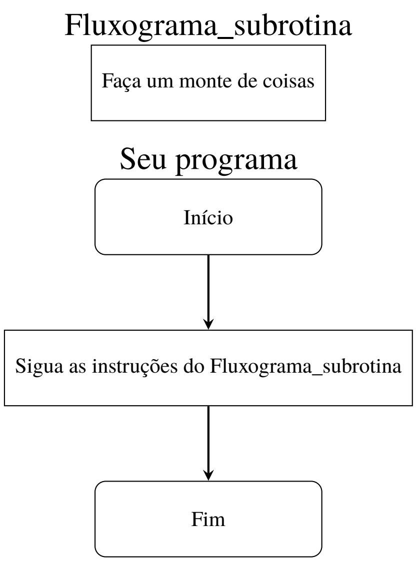

```
enquanto condição
   Processo_caso_verdadeiro()
fim enquanto
```

Ciclo "Para"




```
para estado_inicial ate condição
   Processo_caso_verdadeiro()
fim para
```

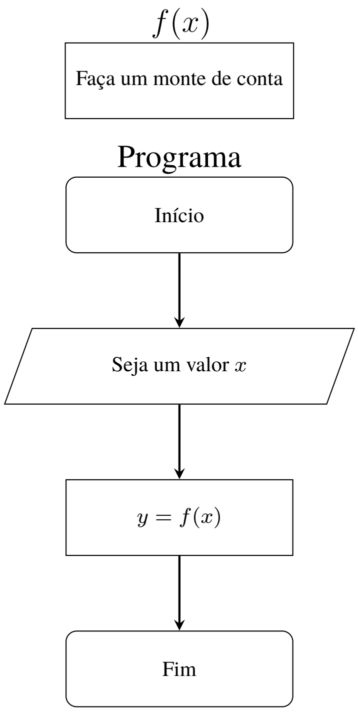
Subrotina




```
subrotina foo
   Faça um monte de coisas
fim foo

inicio
   foo()
fim
```

Função




```
funcao f(x)
   f ← Um monte de conta
fim f

inicio
   variavel x, imagem
   x ← algum valor
   imagem ← f(x)
fim
```

Por exemplo, o pseudocódigo para calcular o fatorial é como mostrado abaixo.

```
inicio
   escrever("Digite um número inteiro")
   ler(n)
   fatorial ← 1
   para i ← 2 ate i=n
      fatorial ← fatorial * i
      i ← i+1
   fim para
   escrever("O fatorial de " n " é " fatorial)
fim
```

Ou, colocando tudo e só o que tem a ver com o fatorial em uma função.

```
funcao fatorial(n)
   tmp ← 1
   para i ← 2 ate i=n
      tmp ← tmp * i
      i ← i+1
   fim para
   fatorial ← tmp
fim fatorial

inicio
   escrever("Digite um número inteiro")
   ler(n)
   escrever("O fatorial de " n " é " fatorial(n))
fim
```

Esse método de calcular o fatorial é o método iterativo, o método recursivo é ainda mais simples de implementar.

```
funcao fatorial(n)
   se n > 1 entao
      fatorial ← n * fatorial(n-1)
   senao
      fatorial ← 1
   fim se
fim fatorial

inicio
   escrever("Digite um número inteiro")
   ler(n)
   escrever("O fatorial de " n " é " fatorial(n))
fim
```

2.3. Material Complementar

1. DIOLINUX. *O que é "o tal do Algoritmo"? - Lógica e Programação*. Disponível em: <https://youtu.be/z1XTcKKRbKM>.
2. ENSICO. *ALGORITMOS DE ORDENAÇÃO*. Disponível em: <https://youtu.be/t9_dHIy0QxA>.
3. ENSICO. *O PROBLEMA DO CAIXEIRO VIAJANTE*. Disponível em: <https://youtu.be/heQeomRPkIw>.
4. ENSICO. *O PODER DA RECURSIVIDADE*. Disponível em: <https://youtu.be/CGi4AkPNlfQ>.
5. ENSICO. *RECURSIVIDADE EM AÇÃO*. Disponível em: <https://youtu.be/6fEbZo7ohYI>.
6. CURSO EM VíDEO. *Curso de Lógica de Programação*. Disponível em: <https://youtube.com/playlist?list=PLHz_AreHm4dmSj0MHol_aoNYCSGFqvfXV>.

3. Alguns Algoritmos

Agora que aprendemos o que são e qual a utilidade dos algoritmos, é bastante conveniente entender alguns algoritmos. Irei apresentar algoritmos famosos que podem ser implementados em qualquer linguagem de programação, de fato no próximo capítulo irei apresentar uma implementação de cada algoritmo mostrado aqui em cada linguagem de programação (serão quatro algoritmos e quatro linguagens).

Os algoritmos serão apresentados em pseudocódigo o que já sugere como implementar em algum programa, além de forçar o aluno a entender, pelo menos, o pseudocódigo.

Uma dica que dou, se você escolhe aprender Fortran, por exemplo, e nos seus estudos consegue entender o meu programa para implementar o Crivo de Eratóstenes, é interessante tentar implementar também os outros algorítmos que irei mostrar.

3.1. Algoritmo de Euclides

Sejam dois números naturais diferentes de zero, se pergunta: *Qual o maior divisor comum (MDC) entre eles?*

O Algoritmo de Euclides é uma maneira de responder essa pergunta de maneira simples e eficaz, sem usar qualquer fatoração (como, creio, aprendemos na escola). O registro mais antigo do uso desse algoritmo é da obra *Elementos*, por volta de 300 a.C., do matemático grego Euclides de Alexandria (300 a.C.), o que faz dele um dos algoritmos mais antigos ainda em uso.

O Algoritmo é baseado no fato de que

$$\mathrm{MDC}(a,b) = \begin{cases} a \text{ se } b = 0 \\ \mathrm{MDC}(b,r) \text{ se } b \neq 0 \text{ e } b < a \end{cases}$$

Onde r é o resto da divisão inteira $\frac{a}{b}$. Note que, na divisão inteira $q = \frac{a}{b}$ temos que $r = a - bq$.

```
funcao MDC(inteiro a, inteiro b) // Considerando b < a

   se b = 0 entao
```

```
      MDC ← a
   senao
      inteiro r
      r ← a - b * (a / b)
      MDC ← MDC(b, r)
   fim se

fim MDC
```

Note que esta é uma definição recursiva do MDC. A partir dela podemos obter a versão iterativa notando que o primeiro argumento sempre é trocado pelo segundo e o segundo sempre é trocado pelo resto antes da próxima iteração.

```
funcao MDC(inteiro a, inteiro b) // Considerando b < a

   enquanto b ≠ 0
      r ← a - b * (a / b)
      a ← b
      b ← r
   fim enquanto
   MDC ← a

fim MDC
```

Os algoritmos recursivos são muito interessantes pela sua simplicidade, também acredito que são mais fáceis de entender. Entretanto, recomendo que sempre que possível implemente a versão iterativa do algoritmo.

3.2. O Crivo de Eratóstenes

Seja um inteiro positivo n, se pergunta: *Quais são todos os números primos menores ou iguais a* n*?*

O algoritmo mais famoso para resolver esse problema é atribuído ao matemático grego Eratóstenes de Cirene (285-194 a.C.) na obra *Introdução à Aritmética* do também grego Nicómaco de Gerasa (60-120 d.C.).

O Crivo de Eratóstenes consiste em escrever uma lista de todos os números inteiros de 2 até n, aceitar que 2 é primo e riscar da lista todos os números múltiplos de 2 e maiores que 2, aceitar que o próximo número não riscado p_2 é primo e riscar todos os múltiplos de p_2 maiores que p_2, ..., aceitar que o próximo número não riscado p_i é primo e riscar todos os múltiplos de p_i maiores que p_i, ..., e assim até que não restem números para riscar.

```
subrotina crivo_de_eratostenes(inteiro n)

   lista[n] // Lista com n elementos
```

```
   para i ← 2 ate n
      lista[i] = i // Escrevendo a lista
      i ← i + 1
   fim para

   para i ← 2 ate n
      se lista[i] não está riscado entao
         lista[i] é primo
         para j ← i + i ate n // Ciclo para riscar os números não primos
            riscar lista[i]
            j ← j + i
         fim para
      fim se
      i ← i + 1
   fim para

fim crivo_de_eratostenes
```

3.3. Sequência de Fibonacci

A sequência atribuída ao italiano Leonardo Fibonacci (1170-1250 d.C.) é uma sequência onde os termos são a soma dos dois números precedentes, a sequência geralmente é iniciada com 1 e 1. Em termos recursivos podemos definir os números da sequência como sendo tais que

$$F_1 = 1$$
$$F_2 = 1$$
$$F_i = F_{i-1} + F_{i-2} \, \forall \, i \geq 3$$

```
subrotina sequencia_de_fibonacci(inteiro n)

   numero_menos1 ← 1
   numero_menos2 ← 1

   escrever("A Sequencia de Fibonacci é:")
   escrever(numero_menos1)
   escrever(numero_menos2)

   para i ← 3 ate n
      numero ← numero_menos1 + numero_menos2
      escrever(numero)
      numero_menos2 ← numero_menos1
      numero_menos1 ← numero
      i ← i + 1
   fim para

fim crivo_de_eratostenes
```

3.4. Torre de Hanói

Se diz que no centro do universo há um templo Hindu onde Brama haveria colocado uma torre com 64 discos de ouro, de raios todos diferentes, e mais duas torres sem discos. Brama ordenou que todos os dias os monges trocassem um disco de torre sem colocar um disco maior sobre um menor. Quando todos os discos estiverem em uma torre diferente então o templo desmoronaria e o mundo desapareceria.

O problema foi desenvolvido pelo matemático francês Édouard Lucas (1842-1891 d.C.), consiste em obter uma sucessão de passos para mover todos os discos de torre e ficou conhecido como problema da Torre de Hanói. Vamos considerar o problema com n discos a desenvolver a solução recursiva apresentada pela Inês. [5] Para tanto considere que todos os n discos se encontram no pilar inicial, que chamaremos de `ini`, que queremos mover todos para o pilar final `fin` e que o outro pilar é o auxiliar `aux`.

Represente por `ini -> fin` o ato de mover um disco da torre inicial para a torre final, etc.

Para o caso onde $n = 2$ a solução é simples, a saber `ini -> aux`, `ini -> fin`, `aux -> fin`.

Note que para mover n discos para o pilar final basta saber mover $n - 1$ discos, dessa forma:

1. Movemos os $n - 1$ discos para a torre auxiliar;
2. Movemos o maior disco para o pilar final; e
3. Voltamos a mover os $n - 1$ discos para o pilar final.

```
funcao torre_de_hanoi(n_discos, ini, aux, fin)

   se n_discos > 0 entao
      torre_de_hanoi(n_discos-1, ini, fin, aux)
      escreva(ini "->" fin)
      torre_de_hanoi(n_discos-1, aux, ini, fin)
   fim se

fim torre_de_hanoi
```

3.5. Material Complementar

1. WIKIPEDIA. *Algoritmo de Euclides*. Disponível em: <https://pt.wikipedia.org/wiki/Algoritmo_de_Euclides>.
2. WIKIPEDIA. *Crivo de Eratóstenes*. Disponível em: <https://pt.wikipedia.org/wiki/Crivo_de_Erat%C3%B3stenes>.

3. REDUCáTICA. *Como encontrar números primos: o Crivo de Eratóstenes*. Disponível em: <https://youtu.be/9yMJeKGzN6I>.
4. WIKIPEDIA. *Sequência de Fibonacci*. Disponível em: <https://pt.wikipedia.org/wiki/Sequ%C3%AAncia_de_Fibonacci>.
5. MATHGURL. *A SEQUÊNCIA DE FIBONACCI*. Disponível em: <https://youtu.be/eVbOxWVC_GY>.
6. WIKIPEDIA. *Torre de Hanói*. Disponível em: <https://pt.wikipedia.org/wiki/Torre_de_Han%C3%B3i>.
7. ENSICO. *RECURSIVIDADE EM AÇÃO*. Disponível em: <https://youtu.be/6fEbZo7ohYI>.

4. Linguagens de programação

CHEGAMOS ao momento de aprender a programar e implementar algoritmos. Esse trabalho pode ser realmente difícil e exige muita dedicação, justamente por isso recomendo que comece a estudar antes de se deparar com a matéria no curso de Física. Para aqueles que não possuem a matéria na grade, ainda assim é sempre de muita importância saber programar nos nossos tempos.

Usar uma linguagem de programação é como usar um pseudocódigo, da mesma forma que escrevíamos pseudocódigo, agora vamos escrever **código fonte** para representar as instruções que o computador deve seguir para resolver determinado problema, as instruções devem ser ainda mais precisas — pois o computador é uma máquina extremamente estúpida — e devem seguir religiosamente as regras da linguagem escolhida para programação. Naturalmente, a primeira tarefa é escolher uma linguagem de programação e começar a estudar. A rigor qualquer linguagem serve e falarei de quatro linguagens, a saber, C, Fortran, Java e Python. Quanto à sua utilidade na física recomendo o seguinte uso:

- Use C ou Fortran para fazer simulações que realizam uma quantidade astronômica de cálculos.
- Use Java ou Python para divulgação científica ou para tratamento de dados.

Neste ponto o leitor pode escolhe uma linguagem, ou todas, e avançar para a próxima seção. Se ainda está indeciso recomendo que estude Fortran, que é uma linguagem feita para se usar na Física. Se quiser aprofundar um pouco mais seus conhecimentos em ciência da computação, continue a leitura no próximo parágrafo.

Você já notou que ao baixar alguns programas você deve escolher entre a versão para Linux ou Windows e entre 32 ou 64bit (Firefox é um exemplo) enquanto que outros programas precisam baixar um arquivo somente para qualquer computador (Minecraft é um exemplo)? O primeiro tipo depende do sistema, enquanto que o segundo não e a razão disso pode estar na linguagem que o programa é escrito. O navegador Firefox, por exemplo, é escrito em C enquanto que o Minecraft é escrito em Java. Mas qual a diferença? Primeiramente note que um computador (ou máquina) é bastante estúpido e não é capaz de ler e executar um código em nenhuma linguagem de programação. Isso torna necessário desenvolver um programa capaz de transformar o código

em algo que o computador pode executar, há basicamente três maneiras de fazer isso: 1) O código fonte é convertido em **código objeto**, o código objeto por sua vez deve ser algo que a máquina é capaz de executar. O programa que faz a conversão é chamado de **compilador**; 2) O código fonte é convertido em ***bytecode***, o *bytecode* por sua vez só pode ser executado por um outro programa específico. O programa que faz a conversão também é chamado de compilador e o que executa é chamado de **Máquina Virtual**; 3) O código fonte é interpretado e executado sem que o usuário tenha que se preocupar em como a máquina se torna capaz de executar o programa. O programa que interpreta é chamado de **intérprete**.

C e Fortran são representantes do grupo cujo o produto da compilação é o código objeto. Esse tipo de compilação costuma gerar instruções em bytes de procedimentos que devem ser seguidos pelo processador, claro está que sistemas e processadores diferentes não reconhecerão o mesmo código objeto, de maneira que o código fonte deve ser compartilhado entre as máquinas e recompilado. A vantagem desse tipo de abordagem é a garantia de que as instruções do código objeto são feitas especificamente para seu processador, otimizando assim seu programa. Java é nosso representante do grupo cujo o produto da compilação é um *bytecode*. As instruções em bytes geradas nesse tipo de compilação não podem ser diretamente executadas pelo processador, a Máquina Virtual Java (JVM) é quem executa as instruções, dessa forma um único *bytecode* pode ser executado em qualquer máquina que possua uma JVM instalada. A vantagem desse tipo de abordagem é que fica mais simples de compartilhar o executável e exije menos conhecimento do usuário final para executar, o que torna o Java uma boa ideia para compartilhar programas de divulgação ou pequenos aplicativos. A desvantagem é a necessidade da JVM como intermediário, o que pode prejudicar a eficiência do programa. Já o Python representa o grupo que não precisa ser compilado. Programas assim podem ser facilmente compartilhados e executados em máquinas diferentes, o que faz do Python ótimo para desenvolver programas de divulgação. As funções disponíveis nas bibliotecas do Python também são ótimas para a análise de dados.

Para o uso na física nós iremos querer utilizar algumas funções pré-implementadas — como seno e raiz quadrada, por exemplo — e todas as linguagens de que falei as possuem de maneira padrão, ou seja, o usuário não precisa implementar as principais funções matemáticas, basta usar. Entretanto a experiência mostra que as implementações do C e do Fortran costumam ser deveras mais eficientes que as do Python, outro fator contra o Python é que sua filosofia de programação pode exigir muitas operações que não seriam feitas em um programa equivalente em C ou Fortran, este último contratempo também pode ser encontrado no Fortran em relação ao C.

Nas seções seguintes irei comentar algumas características de cada linguagem, apresentar o programa "Olá Mundo", e um exemplo de implementação de um dos algoritmos citados no capítulo anterior. Este não é um curso de programação e não entrarei em muitos detalhes básicos mas irei fornecer bons tutoriais na internet que espero que lhes sejam úteis.

4.1. C

ola_mundo.c

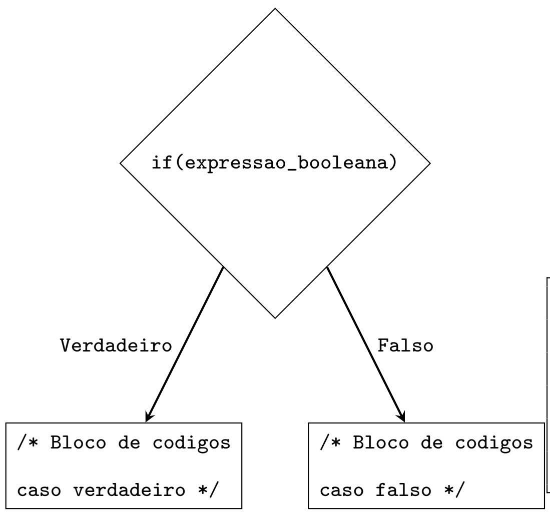
```c
#include <stdio.h>

int main(void){
   printf("Ola Mundo\n");
   return 0;
}
```

Tutoriais

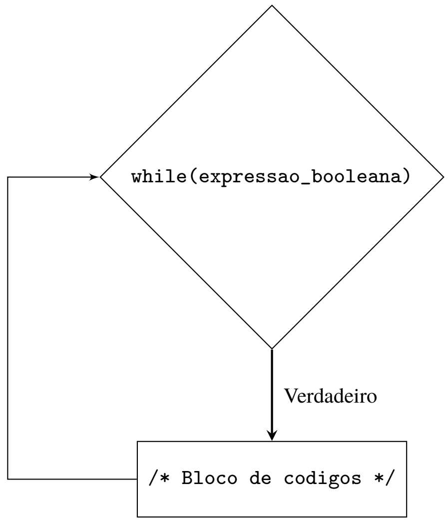
1. EXCRIPT. *Curso de C*. Disponível em: <https://youtube.com/playlist?list=PLesCEcYj003SwVdufCQM
2. DE ALUNO PARA ALUNO. *Linguagem C*. Disponível em: <https://youtube.com/playlist?list=PLa75BYTPDNKZWYypgOFEsX3H2Mg-SzuLW>.
3. PEREIRA, S. do L. *Linguagem C*. [s.n.]. Disponível em: <https://www.ime.usp.br/~slago/slago-C.pdf>.

if



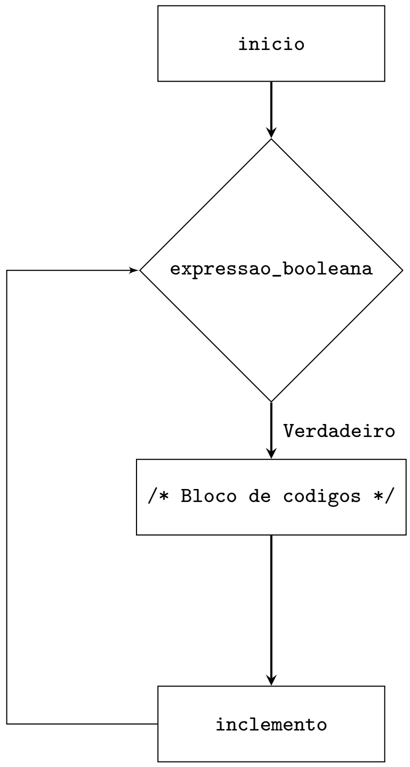

```c
if(expressao_booleana){
   /* Bloco de codigos caso
    verdadeiro */
}else{
   /* Bloco de codigos caso falso */
}
```

while




```c
while(expressao_booleana){
    /* Bloco de codigos */
}
```

for




```c
for(inicio; expressao_booleana;
    inclemento){
    /* Bloco de codigos */
}
```

Cabeçalho

```c
#include <stdio.h> /* Para manusear arquivos */
#include <math.h> /* Para usar funcoes matematicas */
```

mdc.c

```
/* Objetivo: Calcular o MDC entre dois naturais.
   Metodo: Algoritmo de Euclides.
   Argumentos da main: Os dois numeros que se deseja encontar o MDC.
   Autor: I.F.F. dos Santos
   Exemplo de compilacao e execucao no linux:
      gcc mdc.c -o mdc && ./mdc 3 71
*/
int MDC(int a, int b){
   /* Declaracoes */
   int resto;
   /* Comandos executaveis */
   if(a < b) return MDC(b, a); // Para garantir que o 2o argumento eh o menor
   if(b < 0) return -1; // -1 indica erro ao computar o MDC

   while(b != 0){
      resto = a % b;
      /* Iteracao */
      a = b;
      b = resto;
   }
   return a;
}

/* O Algoritmo de Euclides ja termonou,
   o que segue serve somente para testar a funcao MDC() */

#include <stdio.h>
#include <stdlib.h>

int main(int argc, char **argv){
   /* Comandos executaveis */
   printf("MDC(%s, %s) = %d\n", argv[1], argv[2],
   MDC(atoi(argv[1]), atoi(argv[2])));
   return 0;
}
```

4.2. Fortran

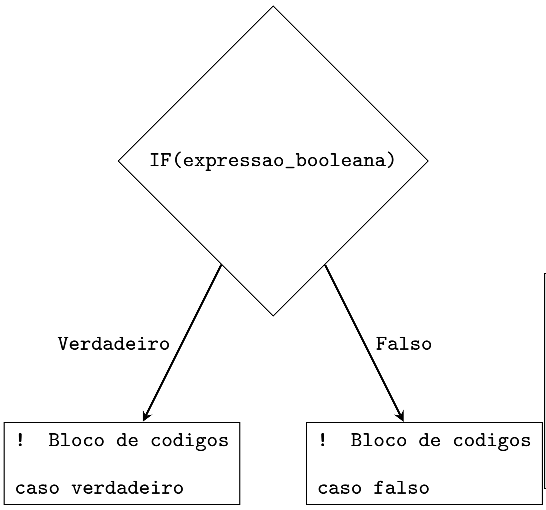
`ola_mundo.f90`

```
PROGRAM ola_mundo
   PRINT *, "Ola Mundo"
ENDPROGRAM
```

Tutoriais

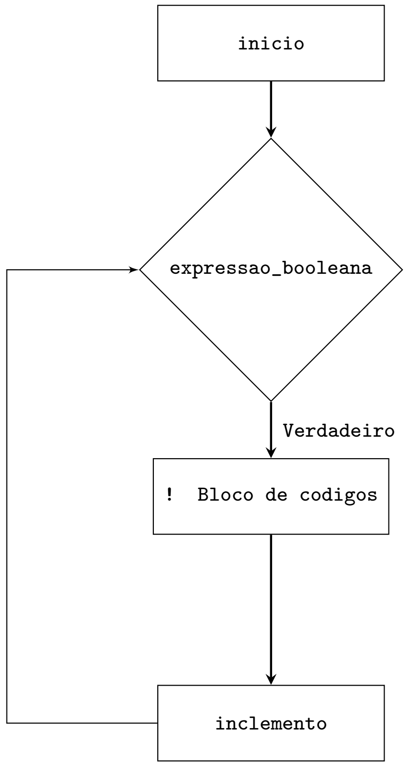
1. WANDERSON FELIPE. *Curso de Fortran 90/95*. Disponível em: <https://youtube.com/playlist?list=PLZGu1IjxtkEj6mcUG6mI8yMhFD16FvVxe>.
2. JAILSSON. *Curso de Fortran*. Disponível em: <https://youtube.com/playlist?list=PLg7DKrtlTwPCOS1
3. DORO, J. V. T. *Introdução à programação em Fortran 90*. [s.n.]. Disponível em: <https://educapes.capes.gov.br/bitstream/capes/206100/2/Introdu%C3%A7%C3%A3o%20%C3%A0%20programa%C3%A7%C3%A3o%20em%20Fortran%2090.pdf>.

IF



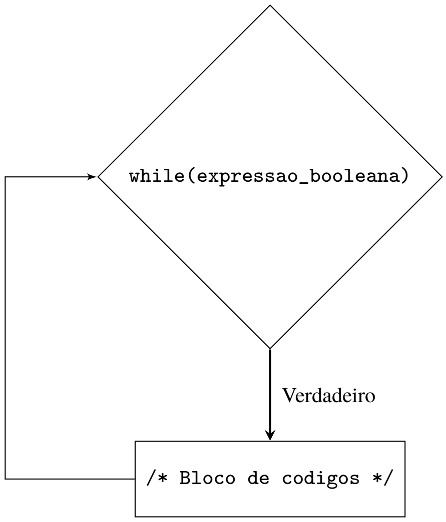

```
IF(expressao_booleana) THEN
   ! Bloco de codigos caso
    verdadeiro
ELSE
   ! Bloco de codigos caso falso
END IF
```

DO




```
DO inicio, expressao_booleana,
    inclemento
   ! Bloco de codigos
END DO
```

do while




```
DO WHILE (expressao_booleana)
   ! Bloco de codigos
END DO
```

crivo.f90

```
!  Objetivo: Calcular todos os numeros primos ate um limite superior.
!  Metodo: Crivo de Eratostenes.
!  Autor: I.F.F. dos Santos
!  Exemplo de compilacao e execucao no linux:
!     gfortran crivo.f90 -o crivo && ./crivo
PROGRAM crivo
   ! O que segue serve somente para testar a rotina crivo_de_eratostenes().
   ! O verdadeiro algoritmo eh a subrotina em CONTAINS.
   ! Declaracoes
   IMPLICIT none
   INTEGER :: limite_superior
   ! Comandos executaveis
   PRINT *, "Bem vindo ao programa Crivo!"
   PRINT *, "Por favor, digite o maior numero da lista onde serao buscados os &
             &numeros primos:"
   READ *, limite_superior
   ! Chamando a rotina para o Crivo
   CALL crivo_de_eratostenes(limite_superior)

CONTAINS
   SUBROUTINE crivo_de_eratostenes(limite_superior)
      IMPLICIT none
      INTEGER, intent(IN) :: limite_superior
      INTEGER :: numero, multiplos
      LOGICAL :: lista(limite_superior)

      ! Os elementos da lista marcados como falsos,
      ! de certeza nao sao primos
      lista = .true.

      PRINT *, "Os numeros primos ate", limite_superior, "sao"

      DO numero = 2, limite_superior
         IF( lista(numero) )THEN
            PRINT *, numero ! Escreve os numeros nao "riscados" da lista
            DO multiplos = numero + numero, limite_superior, numero
               lista(multiplos) = .false. ! "Risca" os multiplos deste da lista
            END DO
         END IF
      END DO
   END SUBROUTINE
END PROGRAM crivo
```

4.3. Java

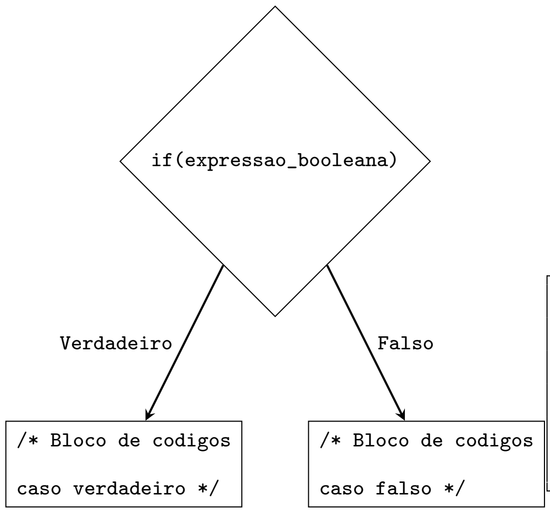
ola_mundo.java

```
class ola_mundo{
    public static void main(String[] args){
        System.out.println("Ola Mundo");
    }
}
```

Tutoriais

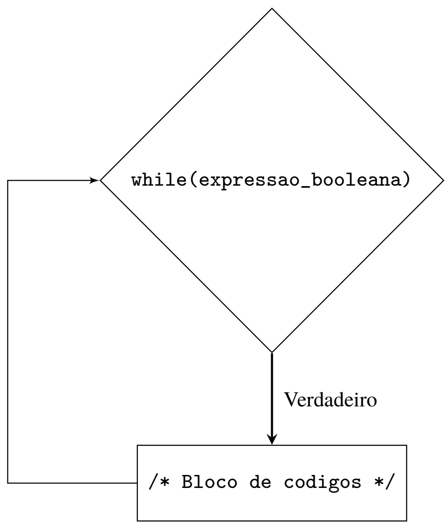
1. CURSO EM VíDEO. *Curso de Java para Iniciantes - Grátis, Completo e com Certificado*. Disponível em: <https://youtube.com/playlist?list=PLHz_AreHm4dkI2ZdjTwZA4mPMxWTfNSpR>.
2. DEVDOJO. *Maratona Java - O maior curso Java em português*. Disponível em: <https://youtube.com/playlist?list=PL62G310vn6nHrMr1tFLNOYP_c73m6nAzL>.
3. CESTA, A. A. *A LINGUAGEM DE PROGRAMAÇÃO JAVA*. [s.n.]. Disponível em: <https://drive.google.com/file/d/1i0mlqAyxR8YKSDKQf1GeXJCuSmjc0UF5/view>.

if



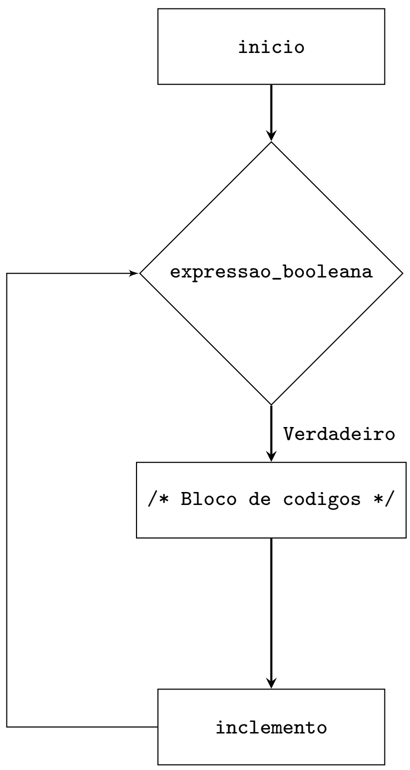

```
if(expressao_booleana){
    /* Bloco de codigos caso
     verdadeiro */
}else{
    /* Bloco de codigos caso falso */
}
```

while




```java
while(expressao_booleana){
    /* Bloco de codigos */
}
```

for




```java
for(inicio; expressao_booleana;
    inclemento){
    /* Bloco de codigos */
}
```

fibonacci.java

```
/* Objetivo: Escrever os N primeiros termos da Sequencia de Fibonacci.
   Metodo: Iteracao F_n = F_{n-1} + F_{n-2}.
   Argumento da main: A quantidade de termos que se deve escrever.
   Autor: I.F.F. dos Santos
   Exemplo de compilacao e execucao:
      javac fibonacci.java && java fibonacci 10
*/
class fibonacci{
   public static void sequencia_de_fibonacci(int quantidade){
      int numero, numero_menos1, numero_menos2;
      numero_menos1 = 1;
      numero_menos2 = 1;
      System.out.println("Sequencia de Fibonacci:\n1\n1");
      for(int i = 3; i <= quantidade; ++i){
         /* Encontre o proximo numero da sequencia */
         numero = numero_menos1 + numero_menos2;
         System.out.println(numero);
         /* Iteracao */
         numero_menos2 = numero_menos1;
         numero_menos1 = numero;
      }
   }
   /* O que segue serve somente para testar
      a rotina sequencia_de_fibonacci() */
   public static void main(String[] args){
      /* Chama a sequencia_de_fibonacci()  usando args[0] como parametro. */
      fibonacci.sequencia_de_fibonacci(Integer.parseInt(args[0]));
      System.exit(0);
   }
}
```

4.4. Python

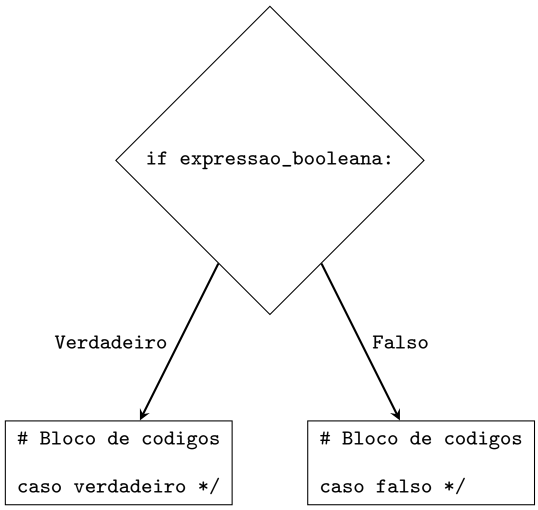
ola_mundo.py

```
print("Ola Mundo")
```

Tutoriais

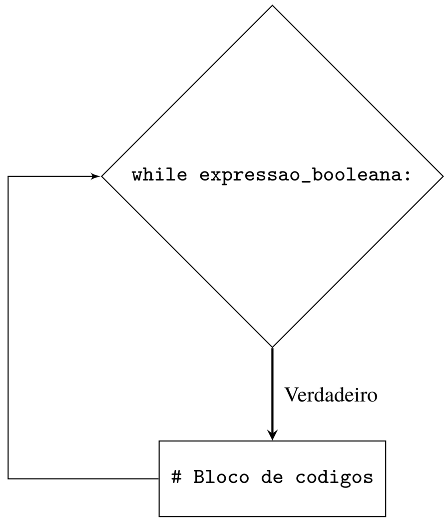
1. CURSO EM VíDEO. *Curso de Python 3 - Mundo 1: Fundamentos*. Disponível em: <https://youtube.com/playlist?list=PLHz_AreHm4dlKP6QQCekuIPky1CiwmdI6>.
2. CURSO EM VíDEO. *Curso de Python 3 - Mundo 2: Estruturas de Controle*. Disponível em: <https://youtube.com/playlist?list=PLHz_AreHm4dk_nZHmxxf_J0WRAqy5Czye>.
3. CURSO EM VíDEO. *Curso de Python 3 - Mundo 3: Estruturas Compostas*. Disponível em: <https://youtube.com/playlist?list=PLHz_AreHm4dksnH2jVTIVNviIMBVYyFnH>.

if




```
if expressao_booleana:
    # Bloco de codigos caso
    verdadeiro
else:
    # Bloco de codigos caso falso
```

while




```
while expressao_booleana:
   # Bloco de codigos
```

hanoi.py

```
#!/usr/bin/env python

# Objetivo: Escrever uma lista com os passos para resolver a Torre de Hanoi.
# Metodo: Recursao.
# Argumento do script: A quantidade de discos.
# Autor: I.F.F. dos Santos
# Exemplo de execucao:
#    python hanoi.py 3

def torre_de_hanoi(n_discos, ini, aux, fin):
   if n_discos > 0:
      # Recursao
      torre_de_hanoi(n_discos-1, ini, fin, aux)
      print(str(ini) + "->" + str(fin))
      torre_de_hanoi(n_discos-1, aux, ini, fin)

# O que segue serve somente para testar a funcao torre_de_hanoi()
import sys
torre_de_hanoi(int(sys.argv[1]), 1, 2, 3)
```

4.5. Material Complementar

1. PROF. FRANCISCO MOURA. *Física Computacional 1*. Disponível em: <https://youtube.com/playlist?list=PLCWOBtltrHovcMBoapQp7NfmsUyu0YRwc>.
2. PROF. FRANCISCO MOURA. *Física Computacional 2*. Disponível em: <https://youtube.com/playlist?list=PLCWOBtltrHovSM2RSLvnKE3oxShD3ksdl>.

Referências

1 DIOLINUX. *O que é "o tal do Algoritmo"? - Lógica e Programação*. Disponível em: <https://youtu.be/z1XTcKKRbKM>.

2 ENSICO. *ALGORITMOS DE ORDENAÇÃO*. Disponível em: <https://youtu.be/t9_dHIy0QxA>.

3 ENSICO. *O PROBLEMA DO CAIXEIRO VIAJANTE*. Disponível em: <https://youtu.be/heQeomRPkIw>.

4 ENSICO. *O PODER DA RECURSIVIDADE*. Disponível em: <https://youtu.be/CGi4AkPNlfQ>.

5 ENSICO. *RECURSIVIDADE EM AÇÃO*. Disponível em: <https://youtu.be/6fEbZo7ohYI>.

6 CURSO EM VíDEO. *Curso de Lógica de Programação*. Disponível em: <https://youtube.com/playlist?list=PLHz_AreHm4dmSj0MHol_aoNYCSGFqvfXV>.

7 WIKIPEDIA. *Algoritmo de Euclides*. Disponível em: <https://pt.wikipedia.org/wiki/Algoritmo_de_Euclides>.

8 WIKIPEDIA. *Crivo de Eratóstenes*. Disponível em: <https://pt.wikipedia.org/wiki/Crivo_de_Erat%C3%B3stenes>.

9 REDUCáTICA. *Como encontrar números primos: o Crivo de Eratóstenes*. Disponível em: <https://youtu.be/9yMJeKGzN6I>.

10 WIKIPEDIA. *Sequência de Fibonacci*. Disponível em: <https://pt.wikipedia.org/wiki/Sequ%C3%AAncia_de_Fibonacci>.

11 MATHGURL. *A SEQUÊNCIA DE FIBONACCI*. Disponível em: <https://youtu.be/eVbOxWVC_GY>.

12 WIKIPEDIA. *Torre de Hanói*. Disponível em: <https://pt.wikipedia.org/wiki/Torre_de_Han%C3%B3i>.

13 EXCRIPT. *Curso de C*. Disponível em: <https://youtube.com/playlist?list=PLesCEcYj003SwVdufCQM5FIbrOd0GG1M4>.

14 DE ALUNO PARA ALUNO. *Linguagem C*. Disponível em: <https://youtube.com/playlist?list=PLa75BYTPDNKZWYypgOFEsX3H2Mg-SzuLW>.

15 PEREIRA, S. do L. *Linguagem C*. [s.n.]. Disponível em: <https://www.ime.usp.br/~slago/slago-C.pdf>.

16 WANDERSON FELIPE. *Curso de Fortran 90/95*. Disponível em: <https://youtube.com/playlist?list=PLZGu1IjxtkEj6mcUG6mI8yMhFD16FvVxe>.

17 JAILSSON. *Curso de Fortran*. Disponível em: <https://youtube.com/playlist?list=PLg7DKrtlTwPCOS10mSuIG04zUrCCmlpXO>.

18 DORO, J. V. T. *Introdução à programação em Fortran 90*. [s.n.]. Disponível em: <https://educapes.capes.gov.br/bitstream/capes/206100/2/Introdu%C3%A7%C3%A3o%20%C3%A0%20programa%C3%A7%C3%A3o%20em%20Fortran%2090.pdf>.

19 CURSO EM VíDEO. *Curso de Java para Iniciantes - Grátis, Completo e com Certificado*. Disponível em: <https://youtube.com/playlist?list=PLHz_AreHm4dkI2ZdjTwZA4mPMxWTfNSpR>.

20 DEVDOJO. *Maratona Java - O maior curso Java em português*. Disponível em: <https://youtube.com/playlist?list=PL62G310vn6nHrMr1tFLNOYP_c73m6nAzL>.

21 CESTA, A. A. *A LINGUAGEM DE PROGRAMAÇÃO JAVA*. [s.n.]. Disponível em: <https://drive.google.com/file/d/1i0mlqAyxR8YKSDKQf1GeXJCuSmjc0UF5/view>.

22 CURSO EM VíDEO. *Curso de Python 3 - Mundo 1: Fundamentos*. Disponível em: <https://youtube.com/playlist?list=PLHz_AreHm4dlKP6QQCekuIPky1CiwmdI6>.

23 CURSO EM VíDEO. *Curso de Python 3 - Mundo 2: Estruturas de Controle*. Disponível em: <https://youtube.com/playlist?list=PLHz_AreHm4dk_nZHmxxf_J0WRAqy5Czye>.

24 CURSO EM VíDEO. *Curso de Python 3 - Mundo 3: Estruturas Compostas*. Disponível em: <https://youtube.com/playlist?list=PLHz_AreHm4dksnH2jVTIVNviIMBVYyFnH>.

25 PROF. FRANCISCO MOURA. *Física Computacional 1*. Disponível em: <https://youtube.com/playlist?list=PLCWOBtltrHovcMBoapQp7NfmsUyu0YRwc>.

26 PROF. FRANCISCO MOURA. *Física Computacional 2*. Disponível em: <https://youtube.com/playlist?list=PLCWOBtltrHovSM2RSLvnKE3oxShD3ksdl>.

A. Instalando C, Fortran, Java e/ou Python no Linux

Para instalar os compiladores de C, Fortran e Java, bem como a JVM e o intérprete de Python no Debian e derivados, como Ubuntu, Linux Mint, etc basta digitar o comando abaixo no terminal.

```
apt update && apt install -y gcc gfortran openjdk-17-jdk python3
```

B. Instalando C, Fortran, Java e/ou Python no Android

A primeira coisa que vamos precisar para programar pelo telefone é de um terminal e o melhor que há é o Termux que pode ser baixado em <https://f-droid.org/packages/com.termux/>. Após baixar e instalar o Termux abra ele e digite as seguintes linhas de comando (tecle enter depois de cada uma).

```
apt update
apt install -y wget gnupg
wget -O - cctools.info/public.key | apt-key add -
echo "deb https://cctools.info termux cctools" >\
$PREFIX/etc/apt/sources.list.d/cctools.list
```

Finalmente, para instalar os compiladores de C, Fortran e Java, bem como a JVM e o intérprete de Python no Android basta digitar o comando abaixo.

```
apt update && apt install -y gcc-cctools openjdk-17 python
```

Para facilitar o uso do `gfortran` digite o seguinte comando no terminal.

```
ln -s ~/../cctools-toolchain/bin/gfortran $PREFIX/bin/gfortran
```

C. Instalando C, Fortran, Java e/ou Python no Windows

Baixe o instalador do OpenJDK (implementação gratis e de código aberto do java) em <https://docs.microsoft.com/pt-br/java/openjdk/download> (procure pelo arquivo de extensão `msi`) e execute ele, certifique-se de marcar a opção *Add to PATH* na janela *Custom Setup*.

Baixe o instalador do Python em <https://www.python.org/downloads/release/python-3100/> e execute ele, lembre-se de marcar a opção *Add Python 3.10 to PATH* antes de começar a instalação. Na última janela pode aparecer a opção *Disable path length limit*, execute-a.

Baixe o MinGW (Pacote que possui o GCC, que possui compiladores de C e Fortran) em <https://sourceforge.net/projects/mingw/> e execute ele, quando aparecer a janela *MinGW Instalation Manager* clique em *Basic Setup* (à esquerda) e depois selecione, ao lado, os pacotes `mingw32-base` e `mingw32-gcc-gfortran` para serem instalados — para fazer isso basta clicar no quadradinho e depois em *Mark for Installation* —. Depois vá à aba *Installation* (em cima) e clique em *Apply Changes*. Uma vez concluida a instalação é preciso adicionar o MinGW ao PATH, para isso pesquise no menu iniciar por `exibir configurações avançadas do sistema` e clique na opção `Variáveis de Ambiente`, procure pela variável `Path` e clique em `editar` e depois em `Novo` e escreva “`C:\MinGW\bin`”.

D. Editores de texto e IDE’s

Para escrever códigos precisamos de editores de texto, em geral queremos editores que deixam as palavras coloridas conforme sua utilidade.

Linux:

- Xed <https://github.com/linuxmint/xed>
- Gedit <https://wiki.gnome.org/Apps/Gedit>

Android:

- DroidEdit <https://play.google.com/store/apps/details?id=com.aor.droidedit>

Windows:

- Gedit <https://wiki.gnome.org/Apps/Gedit>

Uma IDE nada mais é que um editor de texto que possui várias outras funções para facilitar a escrita de códigos, em geral uma IDE é dedicada a uma linguagem específica.

Linux:

- Eclipse <https://www.eclipse.org/downloads/>
- Visual Studio Code <https://code.visualstudio.com/download>
- Code::Blocks (C, C++, Fortran) <https://www.codeblocks.org/downloads/binaries/>
- Apache NetBeans <https://netbeans.apache.org/download/index.html>
- Anjuta <https://anjuta.org/>

Android:

- AIDE (C, Java) <https://play.google.com/store/apps/details?id=com.aide.ui>

Windows:

- Eclipse <https://www.eclipse.org/downloads/>
- Visual Studio Code <https://code.visualstudio.com/download>

- Code::Blocks (C, C++, Fortran) <https://www.codeblocks.org/downloads/binaries/>
- Apache NetBeans <https://netbeans.apache.org/download/index.html>

E. Licença

MIT License

Copyright (c) 2021 I.F.F. dos SANTOS

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.